ETH zürich

# Exercise Session 08 – Geometric Algorithms

**Data Structures and Algorithms**

*These slides are based on those of the course, but were adapted and extended by the teaching assistant Adel Gavranović*

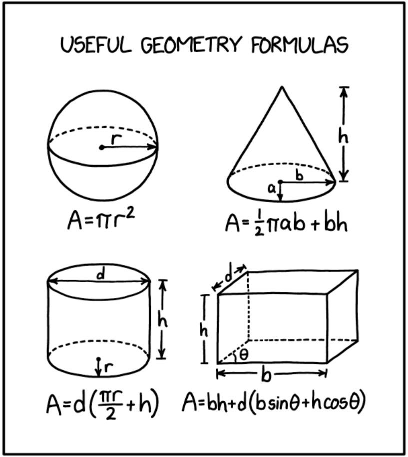
# Today's Schedule

Intro
Follow-up
Feedback for **code** expert
 Closed Hashing
Learning Objectives
Geometric Algorithms
Convex hulls
Sweepline
Geometric Divide & Conquer: Closest Point Pair
Old Exam Question
Outro

n.ethz.ch/~agavranovic

- Exercise Session Material
- Adel's Webpage
- Mail to Adel

# Comic of the Week




▸ xkcd

# 1. Intro

# Intro

- May 9th (Thu) is a public holiday
- We still want to provide a session for you
- Please indicate your availabilities:

▸ when2meet

# 2. Follow-up

# Functors, Lambdas and `const`

**Functors, Lambdas and `const`** (slide 24 ff. from ES07)

- Even though Functors, Lambdas and the keyword `const` are exam relevant (and important concepts!) the details of how exactly w.r.t. `const` a functor converts the arguments passed through its capture (`[]`) are *not exam relevant*
- The reason the function that was marked `const` was able to modify the referenced (&) `unsigned int` that was passed through its capture (`[&count]`), was that it's a reference to a (and not a "real") variable
- The rabbit hole goes even deeper: C++ handles pointers in a very similar (maybe even same?) way

# Functors, Lambdas and `const` – Code

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <functional>

// the filter-helper
template <class Functor>
class Not
{
public:
    Not(Functor & f) : func(f) {}

    template <typename ArgType>
    bool operator()(ArgType & arg) {return !func(arg);}

private:
    Functor & func;
};

// ...
```

```cpp
// ...

// the filter
template<typename T, typename B>
T filter(T list, B pred) {
    T ret;
    std::remove_copy_if(
        list.begin(),
        list.end(),
        std::back_insert_iterator<T>(ret),
        Not<B>(pred)
    );

    return ret;
}

// ...
```

# Functors, Lambdas and `const` – Code

```cpp
// ...

// the functor
class lambda1 {
  unsigned& count;
  int min;
public:
  lambda1(unsigned& c, int m):
    count(c), min(m) {}
  bool operator()(int e) const {
    ++count;
    return min <= e;
  }
};

// ...
```

```cpp
// ...

int main() {

  unsigned count = 0;
  int min = 3;
  std::vector<int> data = {4,-2, 0, 10, 1, 2, 3, 5};
  data = filter(data, lambda1(count, min));

  for (auto datum : data){
    std::cout << datum << " ";
  }

  return 0;
}
```

# 3. Feedback for **code** expert

# Task "Closed Hashing"

- Recap urgent and important, since only 6/24 people got 3/3!

# Double Hashing (cf. Lecture Notes)

For creating a probing sequence that does neither suffer from primary clustering nor from secondary clustering, we can use probe using *double hashing*. We use two hash functions $h(k)$ and $h'(k)$ and *probe along multiples* of $h'(k)$ starting from $h(k)$ , thus[1]

$$S(k) = \underbrace{(h(k) + 0h'(k),\ h(k) + 1h'(k),\ h(k) + 2h'(k),\ h(k) + 3h'(k), \ldots)}_{\text{Probing Sequence (but without the modulo)}} \bmod m$$

[1] where $j$ goes from $0$ (no collision) to $m$ (the size of the hash table)

# Closed Hashing from **code** expert

**TASK** Insert the keys

$$17, 6, 7, 8, 11, 28, 21, 20$$

in this order into an initially empty hash table of size 11. Use open addressing with the hash function

$$h(k) = k \bmod 11$$

and resolve the conflicts using *double hashing* with

$$h'(k) = 1 + (k \bmod 9)$$

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

# Closed Hashing from **code** expert (Solution)

**TASK** Insert the keys

$$17, 6, 7, 8, 11, 28, 21, 20$$

in this order into an initially empty hash table of size 11. Use open addressing with the hash function

$$h(k) = k \bmod 11$$

and resolve the conflicts using *double hashing* with

$$h'(k) = 1 + (k \bmod 9)$$

| 11 | | 6 | 21 | | | 17 | 7 | 8 | 20 | 28 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

# Questions?

# Questions regarding **code** expert from your side?

# 4. Learning Objectives

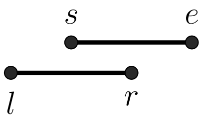
# Learning Objectives

- ☐ Be able to apply simple *hashing methods* by hand
- ☐ Understand how the presented *geometric algorithms* work
- ☐ Understand why the presented *geometric algorithms* work

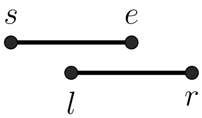
# 5. Geometric Algorithms

# Overlaps of two intervals

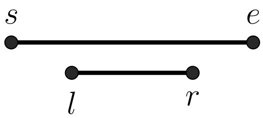
Two intervals $(l, r)$ and $(s, e)$ overlap if




$l \leq s < r$




$l < e \leq r$




or $s \leq l \leq r \leq e$

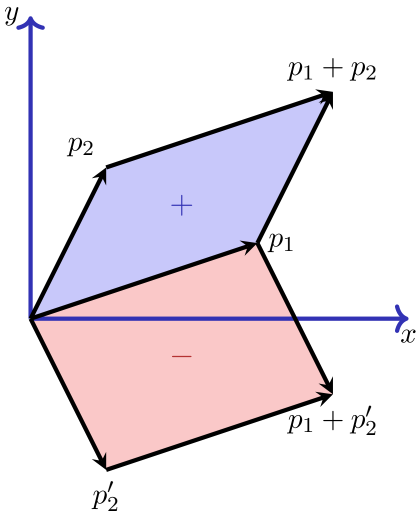
$\Rightarrow$ We can check in constant time whether two intervals intersect.

# Properties of line segments

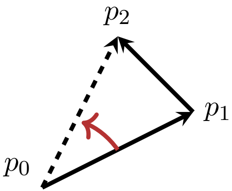
Cross-Product of two vectors $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ in the plane

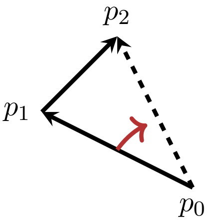
$$p_1 \times p_2 = \det \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix} = x_1 y_2 - x_2 y_1$$

Signed area of the parallelogram



# Turning direction



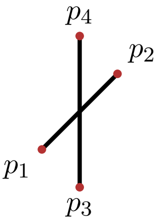




to the left:
$(p_1 - p_0) \times (p_2 - p_0) > 0$

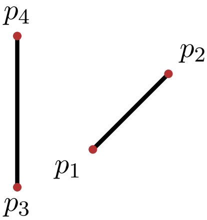
to the right:
$(p_1 - p_0) \times (p_2 - p_0) < 0$

# Intersection of two line segments

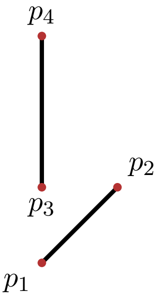
How to figure out whether two segments are intersecting without actually computing the intersection points (division!)?



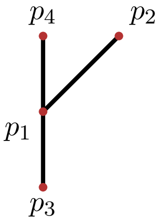

Intersection: $p_1$ and $p_2$ opposite w.r.t $\overline{p_3p_4}$ and $p_3$, $p_4$ opposite w.r.t. $\overline{p_1p_2}$




No intersection: $p_1$ and $p_2$ on the same side of $\overline{p_3p_4}$



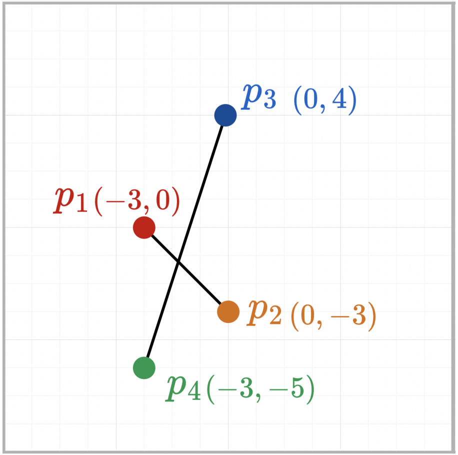

No intersection: $p_3$ and $p_4$ on the same side of $\overline{p_1p_2}$




Intersection: $p_1$ on $\overline{p_3p_4}$

# Intersection of two line segments

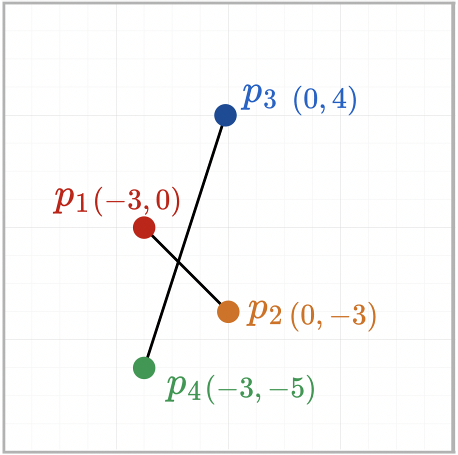
**Part (a)**




Intersection or no intersection?

# Intersection of two line segments

**Part (a)**



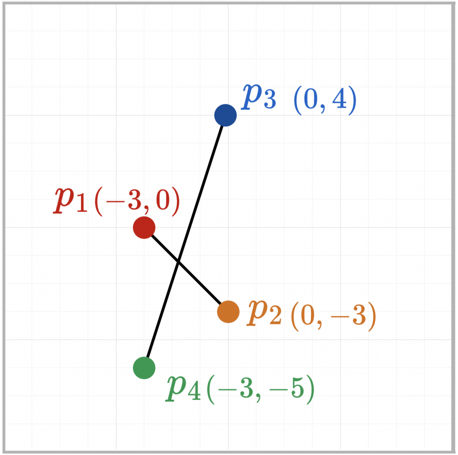

Intersection or no intersection?

**Intersection**

$p_1$, $p_2$ are opposite w.r.t $\overline{p_4 p_3}$,
and $p_3$, $p_4$ are opposite w.r.t. $\overline{p_1 p_2}$.

# Intersection of two line segments

## Part (a)

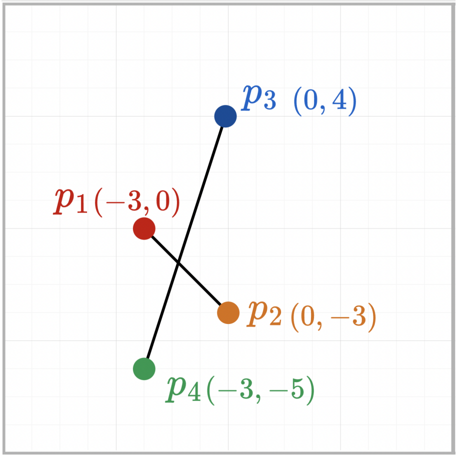



$$(p_3 - p_4) \times (p_1 - p_4) = $$
$$= ((0,4) - (-3,-5)) \times ((-3,0) - (-3,-5)) = (3,9) \times (0,5) = \det \begin{bmatrix} 3 & 0 \\ 9 & 5 \end{bmatrix}$$
$$= (3)(5) - (0)(9) = \mathbf{15} > \mathbf{0}.$$

$$(p_3 - p_4) \times (p_2 - p_4) = $$
$$= ((0,4) - (-3,-5)) \times ((0,-3) - (-3,-5))$$
$$= (3,9) \times (3,2) = \det \begin{bmatrix} 3 & 3 \\ 9 & 2 \end{bmatrix}$$
$$= (3)(2) - (3)(9) = -\mathbf{21} < \mathbf{0}.$$

# Intersection of two line segments

**Part (a)**




and $(p_2 - p_1) \times (p_3 - p_1) =$

$$= ((0,-3)-(-3,0)) \times ((0,4)-(-3,0))$$

$$= (3,-3) \times (3,4) = \det\begin{bmatrix} 3 & 3 \\ -3 & 4 \end{bmatrix}$$

$$= (3)(4) - (3)(-3) = \mathbf{21} > \mathbf{0}.$$

$(p_2 - p_1) \times (p_4 - p_1) =$

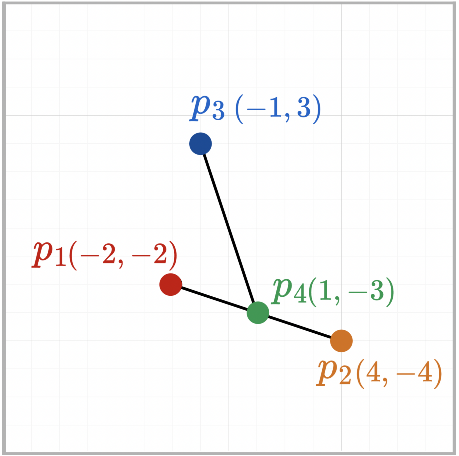
$$= ((0,-3)-(-3,0)) \times ((-3,-5)-(-3,0))$$

$$= (3,-3) \times (0,-5) = \det\begin{bmatrix} 3 & 0 \\ -3 & -5 \end{bmatrix}$$

$$= (3)(-5) - (0)(-3) = -\mathbf{15} < \mathbf{0}.$$

# Intersection of two line segments

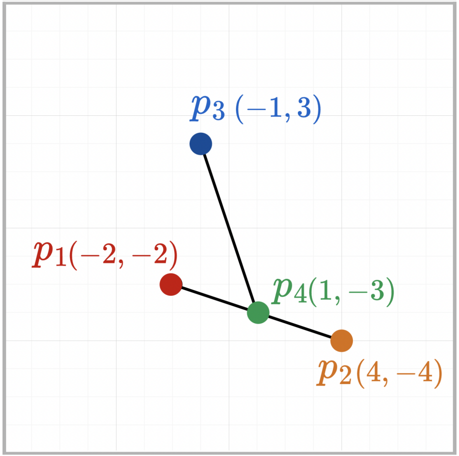
**Part (b)**




Intersection or no intersection?

# Intersection of two line segments

## Part (b)



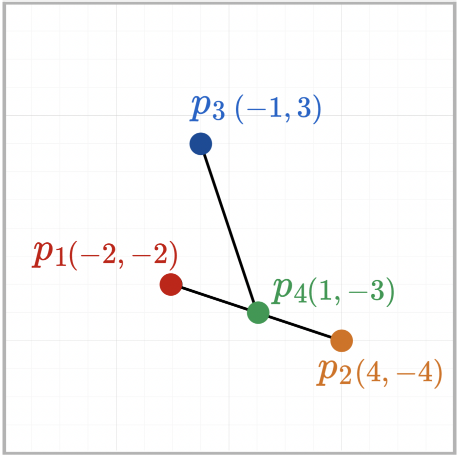

Intersection or no intersection?

**Intersection**

$p_4$ is on $\overline{p_1 p_2}$ for two reasons.

# Intersection of two line segments

**Part (b)**




First,

$(p_2 - p_1) \times (p_4 - p_1) =$

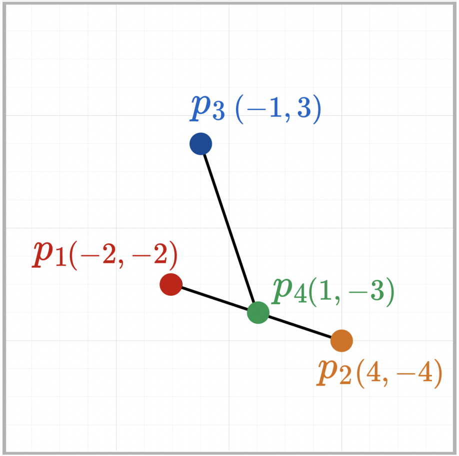
$= \ ((4, -4) \ - \ (-2, -2)) \ \times \ ((1, -3) \ - (-2, -2))$

$= (6, -2) \times (3, -1) = \det \begin{bmatrix} 6 & 3 \\ -2 & -1 \end{bmatrix}$

$= (6)(-1) - (3)(-2) = \mathbf{0}.$

# Intersection of two line segments

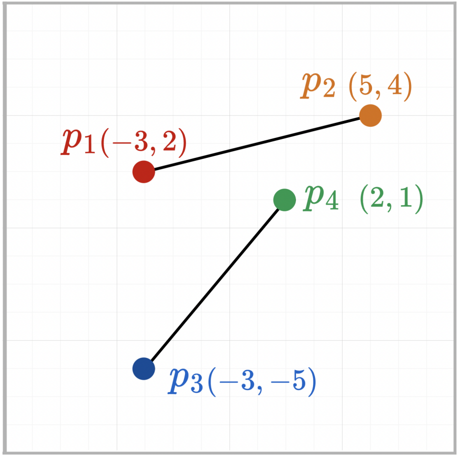
## Part (b)




But this only shows that $p_4$ is in the line created by $\overline{p_1 p_2}$.
To conclude that $p_4$ is in $\overline{p_1 p_2}$, note that
$-2 = p_1[0] \leq 1 = p_4[0] \leq 4 = p_2[0]$
and
$-4 = p_2[1] \leq -3 = p_4[1] \leq -2 = p_1[1]$.

# Intersection of two line segments

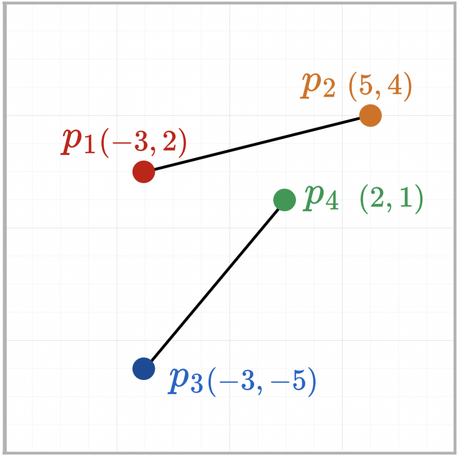
**Part (c)**




Intersection or no intersection?

# Intersection of two line segments

**Part (c)**



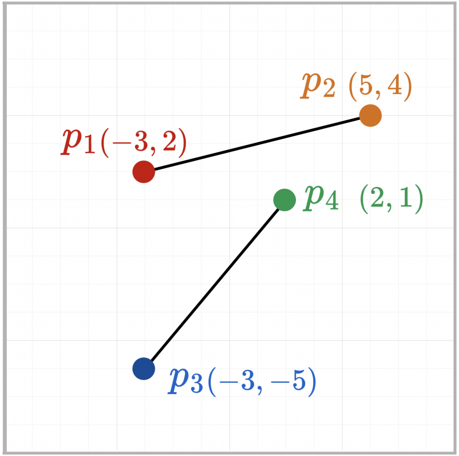

Intersection or no intersection?

**No Intersection**

$p_3$ and $p_4$ are on the same side of $\overline{p_1 p_2}$.

# Intersection of two line segments

**Part (c)**

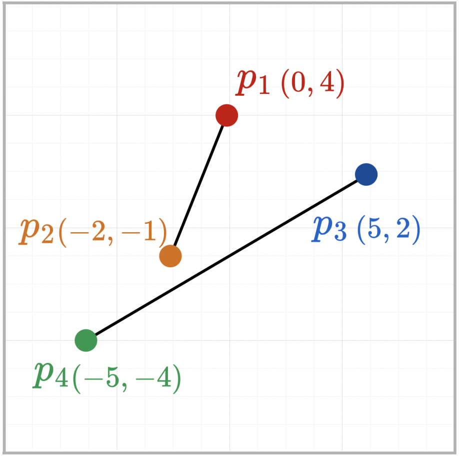



$$(p_2 - p_1) \times (p_3 - p_1) =$$
$$= ((5,4) - (-3,2)) \times ((-3,-5) - (-3,2))$$
$$= (8,2) \times (0,-7) = \det \begin{bmatrix} 8 & 0 \\ 2 & -7 \end{bmatrix}$$
$$= (8)(-7) - (0)(2) = -\mathbf{56} < \mathbf{0}.$$

$$(p_2 - p_1) \times (p_4 - p_1) =$$
$$= ((5,4) - (-3,2)) \times ((2,1) - (-3,2))$$
$$= (8,2) \times (5,-1) = \det \begin{bmatrix} 8 & 5 \\ 2 & -1 \end{bmatrix}$$
$$= (8)(-1) - (5)(2) = -\mathbf{18} < \mathbf{0}.$$

# Intersection of two line segments

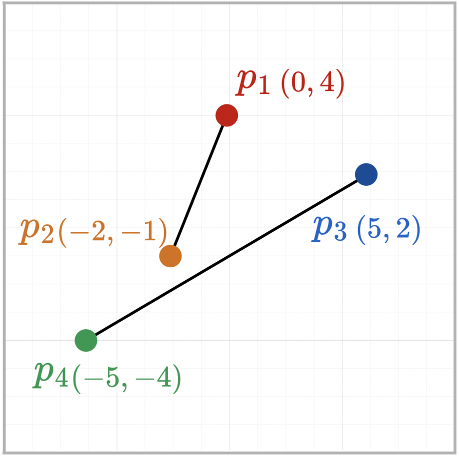
**Part (d)**




Intersection or no intersection?

# Intersection of two line segments

**Part (d)**



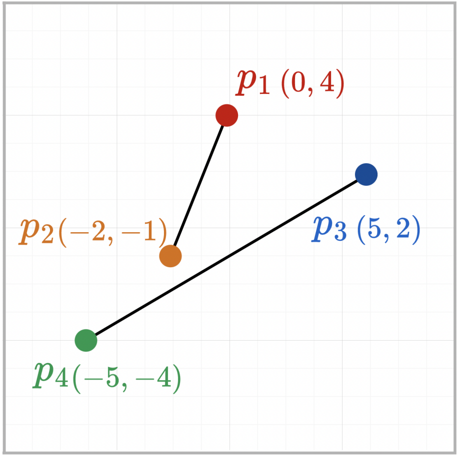

Intersection or no intersection?

**No Intersection**

$p_1$ and $p_2$ are on the same side of $\overline{p_4p_3}$.

# Intersection of two line segments

**Part (d)**




$$(p_3 - p_4) \times (p_1 - p_4) =$$
$$= ((5,2)-(-5,-4))\times((0,4)-(-5,-4))$$
$$= (10,6) \times (5,8) = \det\begin{bmatrix} 10 & 5 \\ 6 & 8 \end{bmatrix}$$
$$= (10)(8) - (5)(6) = \mathbf{60} > \mathbf{0}.$$

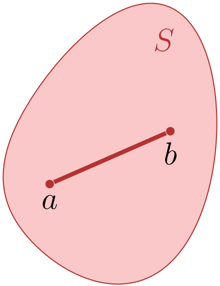
$$(p_3 - p_4) \times (p_2 - p_4) =$$
$$= ((5,2) - (-5,-4)) \times ((-2,-1) - (-5,-4))$$
$$= (10,6) \times (3,3) = \det\begin{bmatrix} 10 & 3 \\ 6 & 3 \end{bmatrix}$$
$$= (10)(3) - (6)(3) = \mathbf{12} > \mathbf{0}.$$

# 6. Convex hulls

# Convex Hull

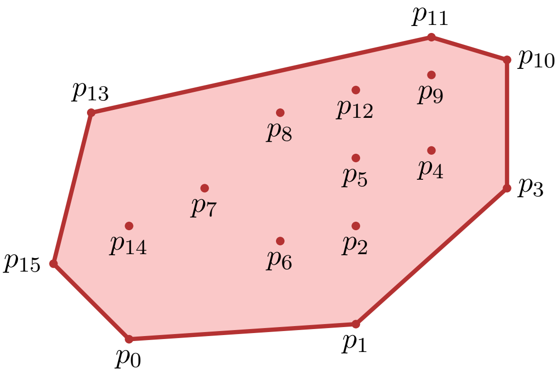
Subset $S$ of a real vector space is called **convex**, if for all $a, b \in S$ and all $\lambda \in [0, 1]$:

$$\lambda a + (1 - \lambda) b \in S$$



# Convex Hull

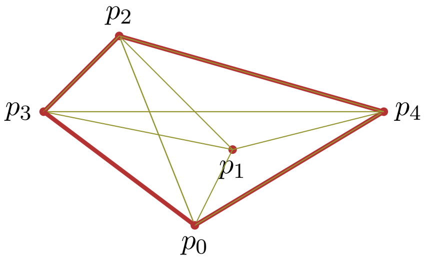
Convex Hull $H(Q)$ of a set $Q$ of points: smallest convex polygon $P$ such that each point of $Q$ is on $P$ or in the interior of $P$.



# Jarvis Marsch / Gift Wrapping algorithm

1. Start with an extremal point (e.g. lowest point) $p = p_0$
2. Search point $q$, such that $\overline{pq}$ is a line to the right of all other points (or other points are on this line closer to $p$.
3. Continue with $p \leftarrow q$ at (2) until $p = p_0$.

# Illustration Jarvis



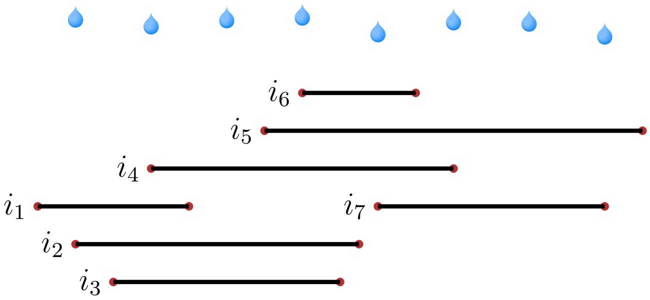

1. Set $H \rightarrow \emptyset$.
2. Find the lowest point $q$.
3. Find the rightmost point $p$, from $q$'s point of view
4. Add $p$ to $H$.
5. Set $q \leftarrow p$ and repeat from step 3 until $q$ is the lowest point
6. $H$ is the convex hull.

# Graham Scan

- Graham Scan: Another algorithm that computes the convex hull
- See the implementation in the lecture slides
- Time complexity:
  - Jarvis March: $\mathcal{O}(h \cdot n)$ where $h$ is the number of corner points on the convex hull
  - Graham Scan: $\mathcal{O}(n \log n)$
- **Question**: When does Jarvis March perform better?
- **Answer**: Jarvis March is better when $h$ is small compared to $n$, as its time complexity depends on the number of corner points on the convex hull.
- Comment: Chan's algorithm improves on both, but is not taught in this course.

# 7. Sweepline

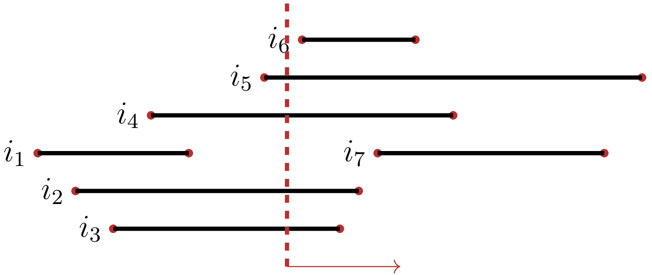
# Preparation: Overlapping Intervals




Questions:

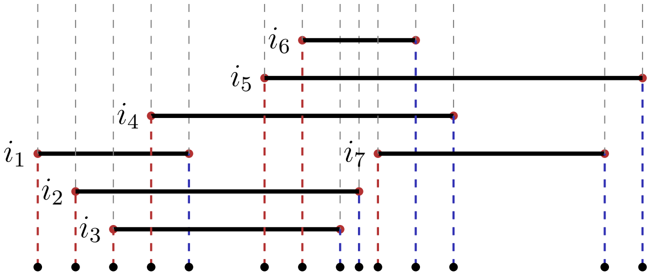
- How many intervals overlap maximally?
- Which intervals (don't) get wet?
- Which intervals are directly on top of each other?

# Preparation: Overlapping Intervals



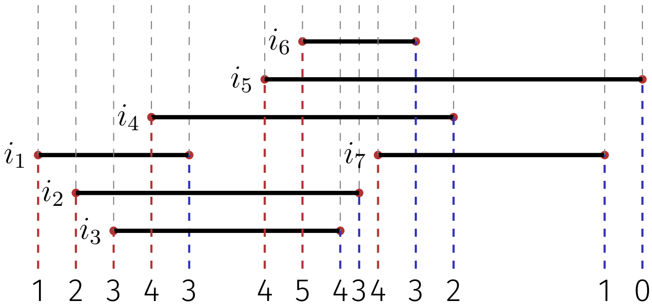

Idea of a sweep line: vertical line, moving in $x$-direction, observes the geometric objects.

# Preparation: Overlapping Intervals




Event list: list of points where the state observed by the sweepline changes.

# Preparation: Overlapping Intervals

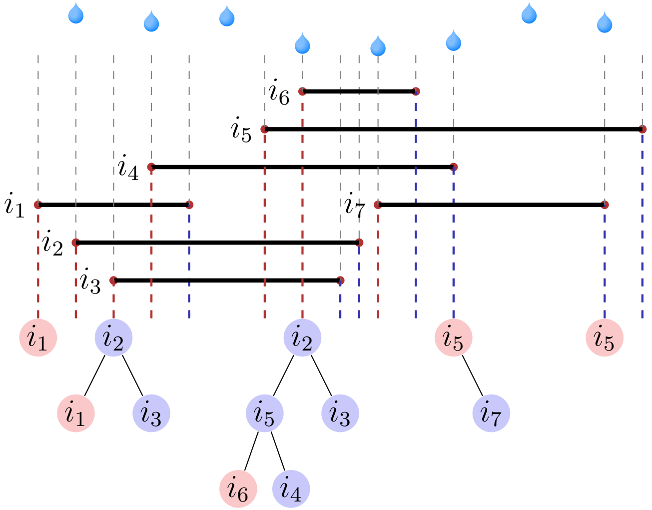



Q: How many intervals overlap maximally?

Sweep line controls a counter that is incremented (decremented) at the left (right) end point of an interval.

A: maximum counter value

# Preparation: Overlapping Intervals

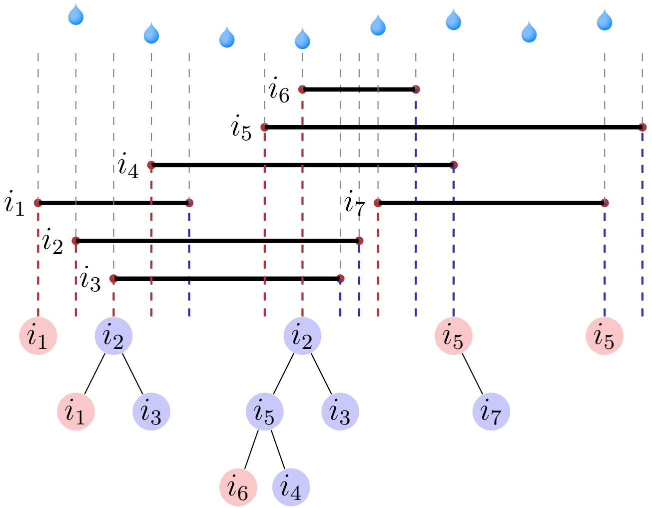



Q: Which intervals get wet?

Sweep line controls a **binary search tree** that comprises the intervals according to their vertical ordering.

A: Intervals on the very left of the tree.

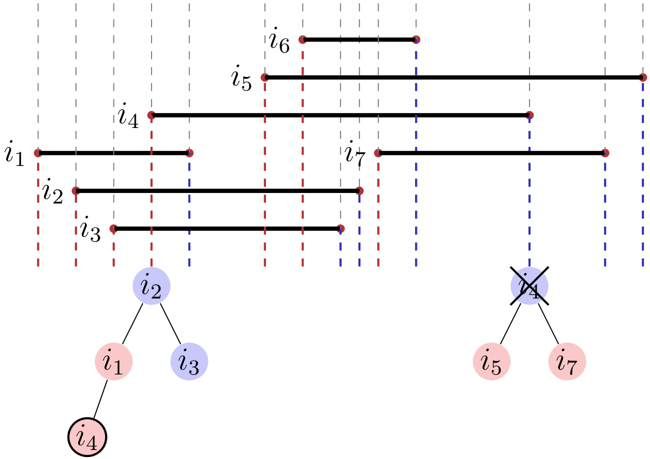
# Preparation: Overlapping Intervals




Q: Why don't we use Max-Heap (instead of BST)?

A: The deletion of an arbitrary element (not the maximum) from a heap is not easy.

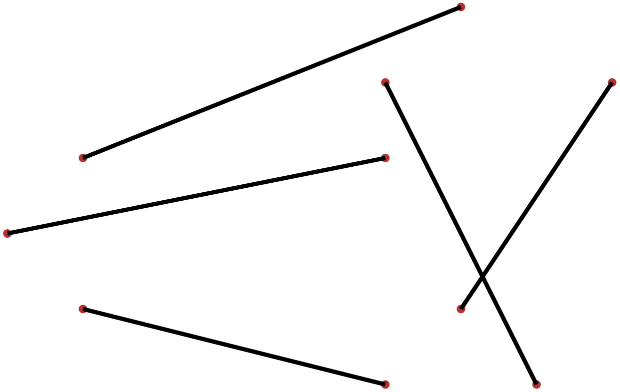
# Preparation: Overlapping Intervals



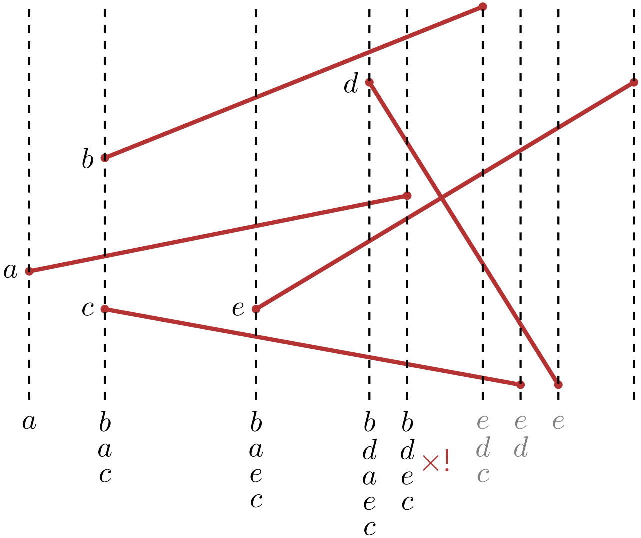

Q: Which intervals are neighbours?

A: If one is the symmetric predecessors or ancestor of the other in the tree.

# Cutting many line segments



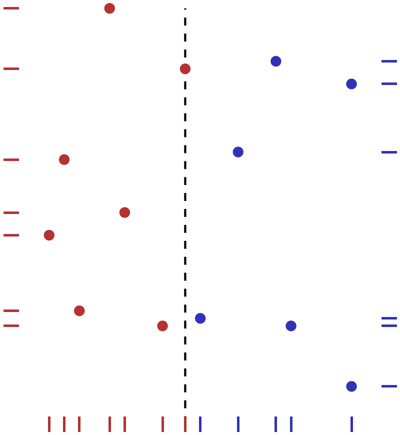
# Intersection of line segments



# 8. Geometric Divide & Conquer: Closest Point Pair

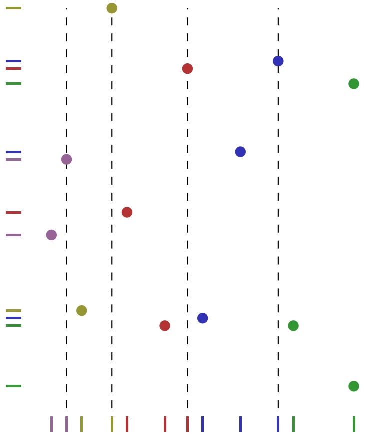
# **Divide** And Conquer: Closest Point Pair

- Set of points $P$, starting with $P \leftarrow Q$
- Arrays $X$ and $Y$, containing the elements of $P$ , sorted by $x$- and $y$-coordinate, respectively.
- Partition point set into two (approximately) equally sized sets $P_L$ and $P_R$, separated by a vertical line through a point of $P$.
- Split arrays $X$ and $Y$ accordingly in $X_L$, $X_R$. $Y_L$ and $Y_R$.



# Divide And **Conquer**: Closest Point Pair

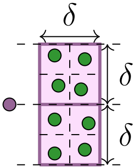
- Recursive call with $P_L, X_L, Y_L$ and $P_R, X_R, Y_R$. Yields minimal distances $\delta_L$, $\delta_R$.
- (If only $k \leq 2$ points: compute the minimal distance directly)
- After recursive call $\delta = \min(\delta_L, \delta_R)$. Combine (next slides) and return best result.



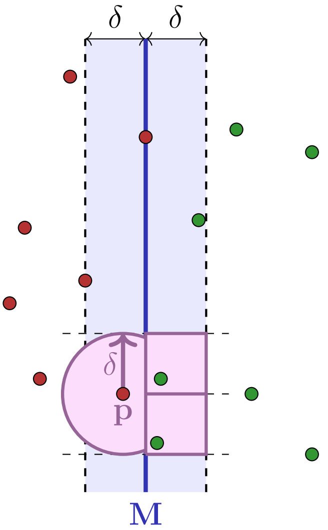
# Minimum Distance across middle line: Observations

Which points are relevant for point $p$?
$\Rightarrow$ the ones in a circle around $p$ with radius $\delta$
**Observation 1:** The relevant points are contained in two $(\delta \times \delta)$-rectangles.

How many points are in these rectangles?
**Observation 2:** At most $8$.




At most one point per $(\delta/2 \times \delta/2)$-rectangle, otherwise they have distance $\sqrt{2} \cdot \frac{\delta}{2} < \delta$.



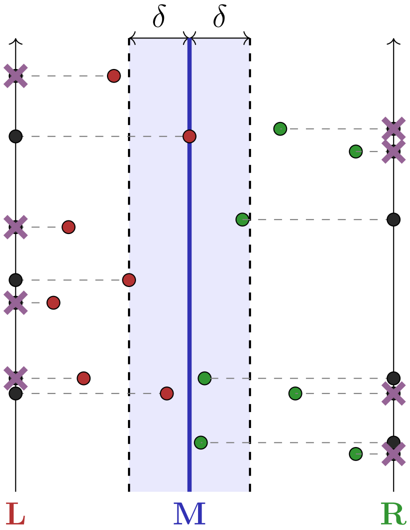
# Minimum Distance across middle line: Algorithm

- sort $L$ and $R$ according to $y$-coordinates
- filter $L$ and $R$ according to band around $M$
- for every remaining point in $L$, compute distance to all points in $R$ in the strip with $y$-distance $\leq \delta$
  $\Rightarrow$ at most $8$ points

**Running time:**

- Sorting: $\Theta(n \log n)$
- Filtering: $\Theta(n)$
- compute the distances: $\Theta(n)$

$\Rightarrow \Theta(n \log n)$ per recursion step



# Implementation

- Goal: recursion equation (runtime) $T(n) = 2 \cdot T(\frac{n}{2}) + \mathcal{O}(n)$.
- Non-trivial: only arrays $Y$ and $Y'$
- Idea: merge reversed: run through $Y$ that is presorted by $y$-coordinate. For each element follow the selection criterion of the $x$-coordinate and append the element either to $Y_L$ or $Y_R$. Same procedure for $Y'$. Runtime $\mathcal{O}(|Y|)$.

Overall runtime: $\mathcal{O}(n \log n)$.

# Questions

- How does the algorithm compare to a brute-force approach?
  - Divide and conquer reduces the problem size at each step, resulting in a time complexity of $\mathcal{O}(n)$, while a brute-force approach has a time complexity of $\mathcal{O}(n^2)$.
- Why do we avoid sorting at each step of the recursion?
  - Sorting is $\mathcal{O}(n \log n)$ and the time complexity of conquer should be linear.

# Questions?

# 9. Old Exam Question

# Traversal

Die Hauptreihenfolgeausgabe eines binären Suchbaumes sei

*Let the pre-order traversal output of a binary search tree be*

$$15, 2, 1, 10, 5, 7, 14, 12.$$

Finden Sie die Nebenreihenfolge.

*Find the post-order traversal.*

| |
|---|
| Nebenreihenfolge / *post-order traversal*: |

# Traversal – Solution

Die Hauptreihenfolgeausgabe eines binären Suchbaumes sei

*Let the pre-order traversal output of a binary search tree be*

$$15, 2, 1, 10, 5, 7, 14, 12.$$

Finden Sie die Nebenreihenfolge.

*Find the post-order traversal.*

Nebenreihenfolge / *post-order traversal*: `1, 7, 5, 12, 14, 10, 2, 15`

# 10. Outro

# General Questions?

# See you next time

Have a nice week!